IN THE CLAIMS

1. (withdrawn) A streaming vector processor comprising:

a plurality of functional units, each functional unit having at least one input and an output, each input being operable to receive an input data value and an associated input data validity tag and the output being operable to provide an output data value and an associated output data validity tag; and

an interconnection switch comprising one or more links, each link operable to couple the output of a functional unit to an input of the at least one input of a functional unit;

wherein the input data validity tag is indicative of the validity of the associated input data value and the output data validity tag is indicative of the validity of the associated output data value.

2. (withdrawn) A streaming vector processor in accordance with claim 1, further comprising at least one data sink coupled to the interconnection switch, wherein the at least one data sink is operable to receive a data value and an associated data validity tag and to commit the data value to memory only if the data validity tag indicates the received data value is valid.

3. (withdrawn) A streaming vector processor in accordance with claim 2, wherein a data sink of the at least one data sink includes a sink iteration counter indicative of the number of remaining data values to be committed, wherein the data value is committed only if the sink iteration counter indicates that there remains at least one data value to be committed.

4. (withdrawn) A streaming vector processor in accordance with claim 3, wherein a data sink of the at least one data sinks is an output stream unit, operable to receive a data value and an associated data validity tag from the interconnection switch and to provide an output data value to a memory

interface only if the data validity tag indicates the received data is valid.

5. (withdrawn) A streaming vector processor in accordance with claim 2, further comprising:

> a controller coupled to and operable to control the interconnection switch, the functional units and the at least one data sink,

wherein the at least one data sink provides a signal to the controller indicative of whether or not all valid data values have been committed to memory.

6. (withdrawn) A streaming vector processor in accordance with claim 1, further comprising:

> an input stream unit coupled to the interconnection switch and operable to retrieve a data value from a data memory and to output the retrieved data value and an associated output data validity tag to the interconnection switch.

7. (withdrawn) A streaming vector processor in accordance with claim 6, wherein the input stream unit includes a source iteration counter indicative of the number of remaining valid data values in the data memory, and wherein the output data validity tag is dependent upon the source iteration counter.

8. (withdrawn) A streaming vector processor in accordance with claim 7, further comprising:

> a controller coupled to and operable to control the interconnection switch, the functional units and the input stream unit,

wherein the input stream unit provides a signal to the controller indicative of whether or not any valid data values remain in the data memory.

9. (withdrawn) A streaming vector processor in accordance with claim 1,

wherein a functional unit of the plurality of functional units includes an output register for storing a data value and an associated data validity tag.

10. (withdrawn) A streaming vector processor in accordance with claim 1, wherein a functional unit of the plurality of functional units includes an input register for storing a data value and an associated data validity tag for each input of the functional unit.

11. (withdrawn) A streaming vector processor in accordance with claim 1, wherein a data value comprises a plurality of sub-words and wherein a data validity tag is associated with each of the plurality of sub-words.

12. (withdrawn) A streaming vector processor in accordance with claim 1, wherein the interconnection switch is re-configurable.

13. (currently amended) A method for executing, on a processor, a pipelined program loop having a loop body but no separately coded prolog instructions for priming the pipeline ~~on a processor~~, the processor comprising a plurality of functional units coupled through an interconnection switch and controlled by a controller, each functional unit of the plurality of functional units having at least one input for receiving an input data value and an associated input data validity tag, the method comprising:

executing the loop body for a plurality of iterations without executing any separately programmed prolog instruction for priming the pipeline; and

at each iteration of [[a]] the plurality of iterations:

determining if the input data values of a functional unit of the plurality of functional units are valid by checking the associated input data validity tags,

executing the iteration whether or not the input data values are valid, and

setting an output data validity tag to indicate that a resulting output data from the functional unit is invalid if any of the input data values to the functional unit is invalid.

14. (currently amended) A method in accordance with claim 13, wherein a functional unit of the plurality of functional units includes a result register for storing an intermediate result and an associated output data validity tag, the method further comprising:

at each iteration of the plurality of iterations:

if all of the input data values are valid, performing a functional operation on the input data values, storing the result of the functional operation in the result register and setting the associated output data validity tag to indicate that the intermediate result is valid.~~; and~~

~~if any of the input data values is invalid, setting the associated output data validity tag to indicate that the intermediate result is invalid.~~

15. (original) A method in accordance with claim 14, further comprising initializing an output data validity tag in the result register of each of the plurality of functional units to indicate that the associated intermediate result is invalid.

16. (original) A method in accordance with claim 15, further comprising:

storing output data values only if the associated output data validity tag indicates that the data is valid.

17. (original) A method in accordance with claim 15, wherein the processor further comprises at least one data source unit, each with an associated source iteration counter, the method further comprising:

initializing each source iteration counter; and at each iteration of the specified number of iterations for which a data value is to be read by a data source unit:

determining from the source iteration counter associated with the data source unit if all data values have been retrieved from memory;

if not all data values have been retrieved from memory, adjusting the source iteration counter, retrieving a data value from a data memory and setting the associated data validity tag to indicate that the result is valid; and

if all data values have been retrieved from memory, setting the output data validity tag to indicate that the result is invalid.

18. (original) A method in accordance with claim 17, further comprising each data source unit signaling the controller when the associated source iteration counter indicates that all data values have been retrieved from memory.

19. (currently amended) A method in accordance with claim 15, wherein the pipelined program loop has no separately coded epilog instructions for draining the pipeline and the processor further comprises at least one data sink, each data sink being associated with a sink iteration counter and operable to receive an output data value and an associated output data validity tag from the interconnection switch, the method further comprising:

initializing the sink iteration counter of each data sink; and

at each iteration of the specified number of iterations:

determining the validity of the output data from the associated output data validity tag;

if the output data is valid:

determining from the sink iteration counter if all data values have been committed to memory;

adjusting the sink iteration counter associated with the data sink unit if not all data values have been committed to memory;

committing the output data value to memory if not all data values have been committed to memory; and

signaling the controller if all data values have been committed to memory, and

if the output data is invalid:
leaving the sink iteration counter unchanged.

20. (currently amended) A method for executing, on a processor, a pipelined program loop having a loop body but no separately coded epilog instructions for draining the pipeline ~~on a processor~~ comprising a plurality of functional units and at least one data sink coupled through an interconnection switch and controlled by a controller, each data sink being associated with a sink iteration counter, the method comprising:

initializing each sink iteration counter;

executing the loop body for a specified number of iterations without executing any separately programmed epilog instruction for draining the pipeline; and

at each iteration of the specified number of iterations for which a data value is to be sunk by a data sink unit:
determining if the sink iteration counter of the data sink unit indicates that all data values have been committed to memory, and

executing the iteration without committing the data value to memory if the sink iteration counter indicates that all data values have been committed to memory.

21. (original) A method in accordance with claim 20, further comprising committing the data value to memory if not all data values have been committed to memory.

22. (original) A method in accordance with claim 20, further comprising adjusting the sink iteration counter associated with the data sink unit if not all data values have been committed to memory.

23. (original) A method as in claim 20, further comprising each data sink unit signaling the controller when the associated sink iteration counter indicates that all data values have been committed to memory.

24. (original) A method as in claim 23, further comprising terminating the execution of the pipelined program loop after all data sink units have signaled to the controller that all their data values have been committed to memory.

25. (new) A method for reducing the number of instructions in a program for controlling a computational pipeline of a processor, the method comprising:

implementing instructions for priming the pipeline as one or more iterations of the loop body;

associating each data value in the pipeline with a data validity tag;

initializing data validity tags to indicate that associated data values are invalid; and

during execution of the loop body:

setting data validity tags to indicate that an associated data value is valid if it is the result of an operation on all valid data; and

setting data validity tags to indicate that an associated data value is invalid if it is the result of an operation on any invalid data.

26. (new) A method in accordance with claim 25, further comprising implementing control words for draining the pipeline as one or more iterations of the loop body.

27. (new) A method in accordance with claim 25, further comprising:
during execution of the loop body:
setting data validity tags to indicate that an associated data value is invalid if an associated source counter has expired.

28. (new) A method in accordance with claim 25, further comprising:
during execution of the loop body:
sinking a data value to a data sink unless an associated data validity tag indicates that the data value is invalid.

29. (new) A method in accordance with claim 25, further comprising:
during execution of the loop body:
sinking a data value to a data sink unless an associated sink counter has expired.